Amendments to the Specification

It is unclear whether the Examiner entered corrections submitted by the Applicants in a Preliminary Amendment dated April 13, 2004. In the Preliminary Amendment, Applicants, among other things, deleted paragraph [0027] of the specification. Thus, the paragraph numbering of the amended specification (as amended by the Preliminary Amendment dated April 13, 2004) is different than the paragraph numbering of the originally filed specification, beginning with paragraph [0027]. The paragraph numbers referred to below are the paragraph numbers of the amended specification.

(1) Please replace paragraph **[0003]** as follows:

[0003] A standard design goal in computer applications is optimal performance. The performance of an application begins with the amount of time the application takes to load into the operating memory of a processing device and prepare for user input. Many applications require time to call methods and functions, and construct objects. This problem occurs in interpreted applications, such as ~~Java~~ JAVA and other common programming languages, as well as non-interpreted applications.

(2) Please replace paragraph **[0006]** as follows:

[0006] The period of time between the command to initiate the program, and the time at which the program commencing processing to external events, depends upon both the time it takes to load the program into memory and the time that it takes to execute initialization code. Any technique that reduces the amount of time spent executing initialization code will decrease the amount of time until the application process external events. For an interactive program such as a desktop application or a rich internet application, this decreases the amount of time between the user event that launches the application (opening a file or choosing a program from a menu on a desktop operating system, or clicking on a link or otherwise requesting the URL that links to the application or an HTML page that embeds it), and the period during which the application is responsive to user events. This is particularly important in environments where code execution is slow, such as virtual machines with bytecode interpreters such as older ~~Java~~ JAVA

implementations and such as the ~~Flash~~ FLASH player. In object-oriented programming languages, bytecode can run on any computer system platform for which a virtual machine or bytecode interpreter is provided to convert the bytecode into instructions that can be executed by the actual hardware processor. Using this virtual machine, the bytecode can optionally be recompiled at the execution platform by a just-in-time compiler. Further, computing the initial state of the application can be time consuming because some of the calculations are complex and because values must sometimes be recomputed.

(3) Please replace paragraph **[0011]** as follows:

[0011] In a further aspect, the method includes reading from a runtime memory space a description of each object of a running application. In a further embodiment, the runtime environment is a virtual machine, and in one particular embodiment, a ~~Flash~~ FLASH renderer. In yet another aspect, the invention includes enumerating a description of each object of a running application using reflection.

(4) Please replace paragraph **[0029]** as follows:

[0030] Figure 6B is a source code listing defining a class included in the application of ~~Figure 7A~~ Figure 6A.

(5) Please replace paragraph **[0032]** as follows:

[0033] The present invention provides a unique method for increasing the speed with which an application can be initialized by a runtime environment. In one aspect, the invention applies to the start-up speed of any application created in a source language and the techniques of the invention can be applied to both interpreted and non-interpreted applications as well. The invention may be used with any runtime environment, including those employing a virtual machine or bytecode interpreter utilized on a processing system. However, a virtual machine or other interpreter is not required. The invention may be implemented with a presentation server and presentation rendering engine in accordance with the teachings of co-pending application serial [[no]] no. 10/092,010, but has applicability to any application.

(6) Please replace paragraph **[0040]** as follows:

[0040] Figure 1 shows a set of application source files and resources 10 available to a compiler 12. The output of the compiler is available to a runtime environment 14 and may be provided responsive to a request for the compiler's output. The various elements can be included within a single processing device, or any combination of multiple processing devices. When provided on separate processing devices, the devices may communicate via network, dedicated connection, wireless connection, or any other connection that is suitable for appropriate communication. The output of the compiler 12 may be provided directly to the runtime environment 14 or stored on a storage device (not shown), and retrieved later by the environment 14. Also shown are an application optimizer engine 16 and a rebuilder engine 18. The optimizer engine 16 may be called from the runtime 14 as a function, or run in a separate process. The output of the compiler 14 is available to the optimizer engine 16, and may be provided directly thereto or retrieved from a storage device. The output of the optimizer 14 is also available to the rebuilder engine 18, as is the output of compiler 12. Both may be provided directly or from stored versions. In general, the optimizer retrieves a description of the application memory state in the runtime 14, and provides it to the rebuilder 18. The rebuilder generates the optimized object code which can be executed in to runtime 14 more rapidly and create the snapshot application state.

(7) Please replace paragraph **[0041]** as follows:

[0042] Figure 2 is a more detailed block diagram of the embodiment of Figure 1 showing how the elements of Figure 1 may be implemented in the context of a presentation system. Figure 2 shows presentation server 50 implemented as a ~~Java~~ JAVA Servlet that compiles server located mark-up language description files and data into object code and hosted in application server 52. In one embodiment, presentation server 50 generates object code for a client presentation renderer 62. The presentation ~~render~~ renderer 62 can be generic software for providing a user-interface or can be specific software for the purpose of communicating with presentation server 50. In one embodiment, client presentation renderer 62 is a Macromedia ~~Flash~~ FLASH Player embedded in a web client as a plug-in. Presentation server 50 can be hosted by

any standard ~~Java~~ JAVA Servlet implementation. When hosted in a J2EE server, the presentation server takes advantage of services that are available including JDBC and JCA. Application Server 52 also includes JDBC to RDBMS services 54, which is in communication with relational database 56. Other types of data sources, other than a relational database can also be used. Presentation server 50 receives requests and sends responses via ~~web server~~ webserver 58, which is in communication with clients via a network. That network can be any standard network known in the art, including the Internet, a LAN, a WAN, etc. For example, Figure 2 shows an HTTP client 60 (e.g. browser) with plug-in 62 (e.g. ~~Flash~~ FLASH Player) in communication with presentation server 50 via web server 58. Also shown in Figure 2 is a rebuilder engine 64 which is provided on the application server 52. The rebuilder engine provides an optimized application to the webserver 58 for delivery to the client 60 when the client makes a request for such an application.

(8) Please replace paragraph **[0043]** as follows:

[0044] Figure 3 shows one embodiment of the method of the present invention. Two general procss stages are shown therein: serializing and rebuilding. Steps 302, 304, 306, 308 and 310 ~~general~~ generally represent the optimizing portion of the process. At step 302, the developed application assets are accessed and compiled in order to create executable code (object file (O1)) for the runtime environment. In one embodiment, this application runtime may be a virtual machine such as the presentation renderer. During this step, each object is marked with a unique identifier (UID) that can be parsed from the object file and is available to the optimization process, as discussed below. At step 304, the runtime execution engine initializes a first memory space (M1) that represents a first program state (S1) which contains built-in objects of the runtime environment.

(9) Please replace paragraph **[0048]** as follows:

[0049] When serialization begins, execution of the user's program is halted. This means that it receives no user-input and no system events while serialization occurs. To serialize the initial state of the program, the serializer enumerates the global values and recursively

enumerates their values. This enumeration is generally referred to as “reflection”. In most contexts, reflection is a term used to describe a program’s ability to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a ~~Java~~ JAVA class to obtain the names of all its members and display them.

(10) Please replace paragraph **[0054]** as follows:

[0054] At step 314, for each object (obj1) in S that has an identifier that refers to an object identified in O1, the object with this identifier in O2 is copied from O2 to O3. In step 316, instructions ~~Instructions~~ are placed in O3 that cause the runtime execution engine to create the same relations between the object (obj3) that O3 causes the runtime execution engine to create, and the other objects (objs3) that O3 causes the runtime to create, that exist between (obj1) and the corresponding objects (objs1) in S2.

(11) Please replace paragraph **[0058]** as follows:

[0058] If the function is not a closure, then the method determines whether a unique function ID (FID) for the function has been established at step 416. In a unique aspect of the present invention, functions associated by an application are uniquely identified. Each function is provided with an (FID) which is associated with the function. An FID data store is maintained on a processing device, such as the presentation server 50, wherein each function is linked to its function ID. When the rebuilding the application, requests for a particular function are retrieved by looking it up in the FID table. If an FID has been established, then the method records the placement of the function relative to the object at step 418. If no FID has been established, then the method creates an FID at step 420 and creates an entry in an FID table and records this entry in the serialization record, in step 422.

(12) Please replace paragraph **[0060]** as follows:

[0061] A portion of one embodiment of a serialized representation created by the process described with respect to Figure 4 is shown in Figure 5. The example shown in Figure 5 is

generated from a Macromedia ~~Flash~~ FLASH runtime executing the output of a presentation server compiled a mark-up language description of a content application. In other embodiments of the invention, other types of source languages may be used. In this mark up language example, an application source file may contain at most one root element. In source application files, an element such as a canvas element defined in co-pending application serial [[no]] no. 10/092,010 defines properties of the global canvas, which is the root of the runtime view hierarchy; all other views are directly or indirectly contained within the canvas element.

(13) Please replace paragraph **[0066]** as follows:

[0067] The serialization process does not prioritize objects during the serialization process. In order to handle all types of runtimes, including a presentation renderer such as ~~Flash~~ FLASH wherein the content of the variable binding application cannot be introspected nor serialized, the serialization process uses single pass through the frozen application, and detaches each and every object as shown in Figure 4.

(14) Please replace paragraph **[0067]** as follows:

[0068] In a further unique aspect, as described above, closures are handled in a different manner. Essentially, closure is a function which calls another function--a pair of a function and a variable binding environment within which the function is executed. The function code is present in the object file (O1). In one embodiment, the variable binding environment is computed at runtime. In the some contexts, such as where the runtime environment is designed for a presentation ~~render~~ renderer such as a ~~Flash~~ FLASH player, the content of the variable binding environment cannot be introspected and therefore cannot be serialized. This therefore puts it in a different category than functions, assets and objects. Functions and assets are represented in the object file, and JavaScript objects are represented in the application state and can be introspected, and therefore both can be serialized. Closures are neither present in the object file nor can they be introspected.

(15) Please replace paragraph **[0078]** as follows:

[0079] A method for decreasing a computer application's start-up time. In one aspect, the method comprises: creating a serialized representation of application objects in a runtime environment; building an object code file using the serialized representation; and providing the application to a new runtime environment. In another embodiment the method may include the steps of: compiling an application provided in ~~an source language~~ a source language; initializing the application in a runtime environment; and creating a serialized representation of the application.